AN214821

Upgrading Human Interface Device Firmware (Example Protocol, Commands, and Scripts)

Associated Part Family: CYW20730/CYW20733

This document describes the human interface device (HID) layer protocol that the Cypress CYW20730 and CYW20733 use to support Bluetooth host-to-HID over-the-air upgrades. In addition, the document provides some sample commands and scripts.

This document is primarily intended for software developers writing Bluetooth host-device applications that will be used to communicate with CYW20730- and CYW20733-based HIDs.

Contents

1 About this Document

1.1 Cypress Part Numbering Scheme

Cypress is converting the acquired IoT part numbers from Broadcom to the Cypress part numbering scheme. Due to this conversion, there is no change in form, fit, or function as a result of offering the device with Cypress part number marking. The table provides Cypress ordering part number that matches an existing IoT part number.

Table 1. Mapping Table for Part Number between Broadcom and Cypress

Broadcom Part Number	Cypress Part Number
BCM20730	CYW20730
BCM20733	CYW20733

1.2 Acronyms and Abbreviations

In most cases, acronyms and abbreviations are defined on first use.

For a comprehensive list of acronyms and other terms used in the documents, go to:

http://www.cypress.com/glossary

2 IoT Resources

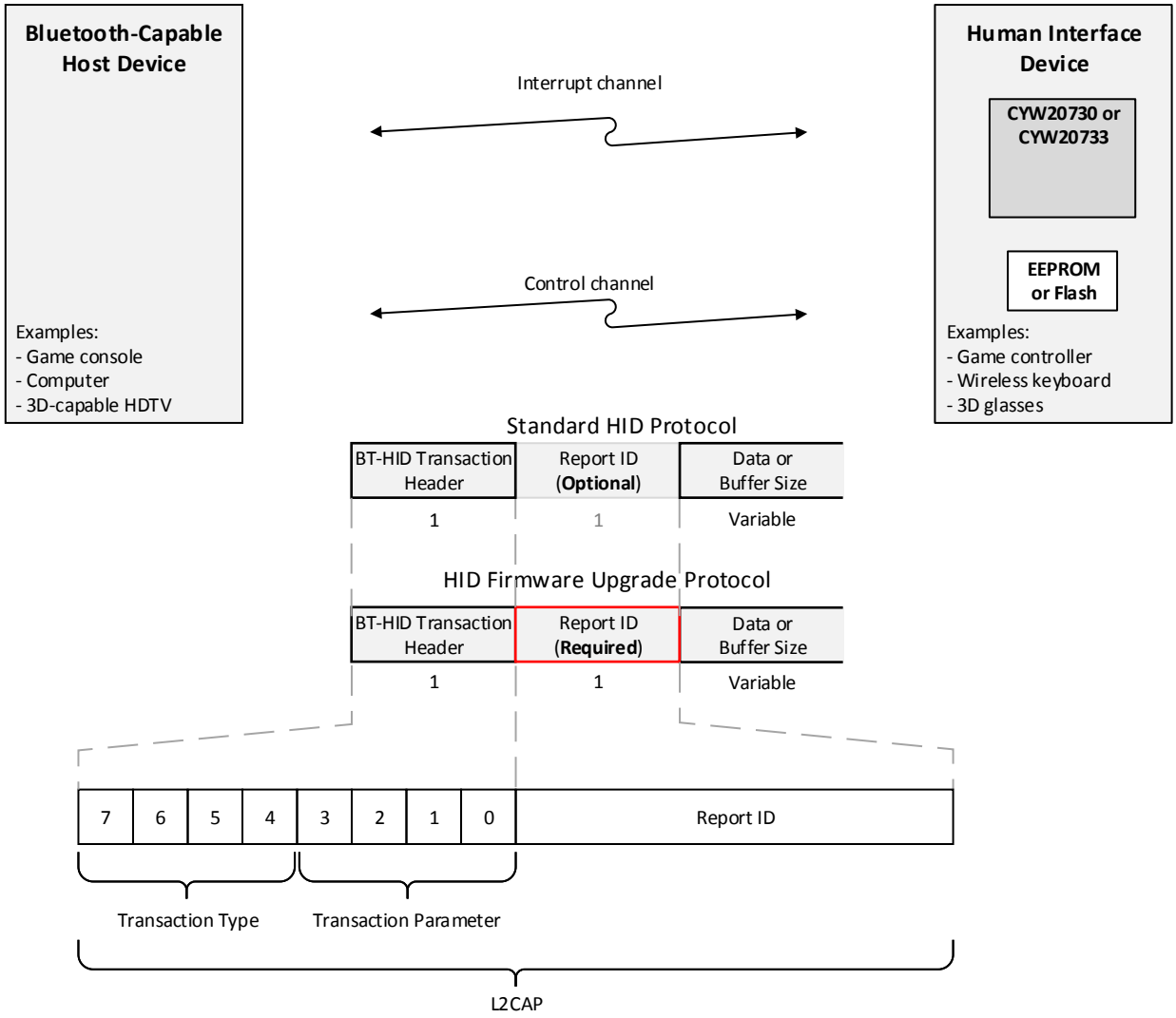
Cypress provides a wealth of data at http://www.cypress.com/internet-things-iot to help you to select the right IoT device for your design, and quickly and effectively integrate the device into your design. Cypress provides customer access to a wide range of information, including technical documentation, schematic diagrams, product bill of materials, PCB layout information, and software updates. Customers can acquire technical documentation and software from the Cypress Support Community website (http://community.cypress.com/).

3 Introduction

A human interface device (HID) provides a means for human data input and output to and from a remote host.

Figure 1 shows an application diagram where a CYW20730 or CYW20733 is used in a HID.

Figure 1. CYW20730 and CYW20733 Application Diagram



The standard Bluetooth HID protocol has an optional report ID byte that follows the transaction header byte when it is used.

The extensions to the standard HID protocol in this document require using the report ID byte.

4 Standard HID Protocol Transaction Types

As shown in Figure 1, all interrupt/control channel messages between a host and a HID begin with a BT-HID transaction header. The transaction header has two 4-bit fields, the transaction type and an associated transaction parameter.

Table 2 provides the complete set of transaction types, and the hexadecimal value and transaction length of each type.

Table 2. Transaction Type Information

Transaction Type	BT-HID Transaction Header Value	Transaction Length (Bytes)[1]	Description
HANDSHAKE	0x0	1	A HID response to all set requests and all get requests where an error is detected.
HID_CONTROL	0x1	1	A control used to request a state change in the Bluetooth HID.
Reserved	0x2–0x3	–	–
GET_REPORT	0x4	1–4	A host request for a report from the HID.
SET_REPORT	0x5	1 + report-data payload	A host report to the HID.
GET_PROTOCOL	0x6	1	A host request for the current protocol.
SET_PROTOCOL	0x7	1	A host set of the HID protocol.
GET_IDLE	0x8	1	A host request for the current idle state.
SET_IDLE	0x9	2	A host set of the idle state.
DATA	0xA	1 + report-data payload	The first (and possibly only) data payload of report-data payloads.
DATC	0xB	1 + continuation of report-data	Subsequent payloads associated with an overall report-data payload that exceeds a negotiated MTU.
Reserved	0xC–0xF	–	–

Note: Of the transaction types in Table 2, GET_REPORT, SET_REPORT, DATA, and HANDSHAKE are the only types that pertain to the content in the remainder of this document.

[1] All transaction lengths include the 1-byte BT-HID transaction header.

5 Transaction Types Supporting a Firmware Upgrade

Note: The transaction format information in this section is from a Bluetooth specification. For complete transaction format information, see *Human Interface Device Profile 1.1* (available from www.bluetooth.org).

5.1 GET_REPORT Transaction Type

This transaction type is used by the host to request a report from the HID. If the request is received without errors, then the HID responds by returning a payload of transaction type DATA on the control channel that contained the requested report. If there are errors, then the HID returns a HANDSHAKE with an error message.

Table 3 describes the byte and bit fields in a GET_REPORT transaction.

Table 3. GET_REPORT Transaction Type Description

Field	Size (Bytes	Bits	Description
Request	1		Request characteristics.
		7:4	Transaction type. Set to 4 to indicate GET_REPORT.
		3	Size 0: The host has allocated a buffer equal to the size of the report. 1: A 2-byte BufferSize field follows the Report ID. This field indicates the size of the buffer allocated by the host. HIDs must limit the returned payload size to BufferSize. The BufferSize must be increased by 1 byte for Boot mode reports in order to include the Report ID imposed by BT-HID.
		2	Reserved. Set to 0.
		1:0	Report type. 0: Reserved 1: Input 2: Output 3: Feature
Report ID	1	7:0	Optional field for standard HID protocol. Required for a HID FW upgrade. Report ID of the requested report (see Table 7).
BufferSize	2	15:0	Optional field. Maximum number of bytes to transfer during the data phase. This field does not exist if the Size field (bit 3 in the request byte) is 0. BufferSize is little endian.

5.2 SET_REPORT Transaction Type

The Bluetooth host uses this transaction type to send a report to a HID. A single report follows either the request or the report ID if a report ID is present. Only one report can be sent per SET_REPORT transaction.

Table 4 describes the byte and bit fields in a SET_REPORT transaction.

Table 4. SET_REPORT Transaction Type Description

Field	Size (Bytes	Bits	Description
Request	1		Request characteristics.
		7:4	Transaction type. Set to 5 to indicate SET_REPORT.
		3:2	Reserved
		1:0	Report type. 0: Reserved. 1: Input. 2: Output. 3: Feature.
Report ID	1	7:0	Optional field for the standard protocol. Required for a HID FW upgrade. Report ID of the payload (see Table 7).
Report Data Payload	n	–	Report data. See “SETUP_READ” for the payload used to set up a data read from the HID during a firmware upgrade.

5.3 DATA Transaction Type

The Bluetooth HID uses this transaction type to send a report to the Bluetooth host.

Table 5 describes the byte and bit fields in a DATA transaction.

Table 5. DATA Transaction Type Description

Field	Size (Bytes	Bits	Description
Request	1		Request characteristics.
		7:4	Transaction type. Set to 10 (0xA) to indicate a DATA transaction.
		3:2	Reserved (0)
		1:0	Report type. 0: Other. 1: Input. 2: Output. 3: Feature. The report type returned from the HID during a firmware upgrade is 0.
Report ID	1	7:0	Optional field for the standard protocol. Required for a HID FW upgrade. Report ID of the payload (see Table 7).
Report Data Payload	n	–	Report data. See “READ” for the payload used while reading data from the HID during a firmware upgrade.

6 HID Firmware Upgrade Protocol

The protocol in this section describes a HID layer upgrade method for upgrading HID firmware (FW).

The host must open one control channel and one interrupt channel before starting a HID firmware upgrade.

In order to support a HID FW upgrade, some previously unused report IDs are assigned. Table 6 shows the assigned report IDs and those that are unused.

Table 6. Bluetooth Report IDs from HID Profile Specification Version 1

Report ID	Device	Report Size (Bytes)
0	Reserved	Not applicable
1	Keyboard	9
2	Mouse	4
3–255 (0x03–0xFF)	Reserved (Unused)	Not applicable

The Cypress approach to supporting a HID FW upgrade uses some of the many unused report IDs.

6.1 Report IDs and Associated Commands

Table 7 describes extended report command types that can be assigned to six of the previously unused report IDs.

Table 7. Report IDs to Support a HID FW Upgrade

Report ID	Report Command Type	Report Size (Bytes)	Transaction Type	Description
0x70	ENABLE_FWU	1	SET_REPORT[2]	Command to enable a FW upgrade.
0x71	SETUP_READ	8	SET_REPORT	Command to set the read address and length for a subsequent READ transaction.
0x72	READ	1 + n	GET_REPORT	Command to read data from the HID.
0x73	ERASE	8	SET_REPORT	Command to erase HID memory.
0x74	WRITE	8 + n	SET_REPORT	Command to write to HID memory.
0x75	LAUNCH	8 + n	SET_REPORT	Command to jump execution to a specific HID memory location.

[2] The variable n represents the number of payload data bytes in the report. It does not include the report ID, address, length, and checksum bytes. The overall size reported in this column includes all bytes except the 1-byte transaction header.

6.2 Typical HID Firmware-Upgrade Packet Structure

Most of the report commands (see Table 7) use the following basic packet structure.

```
struct HIDFWUReportHeader
{
    UINT8     reportID; // Report ID (1 byte)
    UINT32    address;  // Address (4 bytes, little endian)
    UINT16    length;   // Length in bytes (2 bytes, little endian
    UINT8     data[];   // Data bytes (if present, their count must match "length")
    UINT8     checksum; // Checksum (1 byte)
};
```

Table 8 describes the fields of the basic HID firmware-upgrade report header.

Table 8. Basic HID Firmware-Upgrade Report Header

Report ID	Transaction Type	Description
reportID	1	The Report ID of the packet. See Table 7 for a complete list of report IDs.
address	4	The start address of a READ, WRITE, ERASE, or LAUNCH command.
length	2	The number of bytes to be read, written, or erased.
checksum	1	The 2's complement checksum of HIDFWUReportHeader, ignoring carry bits. The checksum calculation starts with the reportID. See 8.6 Calculating a Checksum for sample Perl code.
reportID	1	The Report ID of the packet. See Table 7 for a complete list of report IDs.
address	4	The start address of a READ, WRITE, ERASE, or LAUNCH command.

7 HID Firmware Upgrade Commands

The packet formats and descriptions for the following commands are covered in this section:

- ENABLE_FWU
- SETUP_READ
- READ
- ERASE
- WRITE
- LAUNCH

For reference Perl scripts of the commands in this section, see 8 HID Firmware Upgrade Sample Scripts.

CYW20730 and CYW20733 firmware will verify the total transaction sizes of all commands. In the verification, all transaction bytes will be accounted for, including the transaction header. If the size of a command does not match the expected size, then a HANDSHAKE error with error code ERR_INVALID_PARAMETER will be returned. This same error code will be returned in a HANDSHAKE transaction when the HID detects any error.

7.1 ENABLE_FWU

This command is used to enable a HID firmware upgrade. Upon receiving this command, a watchdog timer will be enabled in the HID.

The ENABLE_FWU command must be sent before sending any ERASE, WRITE, and LAUNCH commands.

The firmware upgrade process cannot be disabled after sending the ENABLE_FWU command. For a successful firmware upgrade, the HID will be restarted before watchdog expiration. For an unsuccessful upgrade, the HID will be restarted when the watchdog timer expires.

The following packet format is used for this command:

```
struct HIDFWUReportHeader
{
    UINT8 reportID; // 0x0, but changeable.
    UINT8 checksum;
};
```

Report Data Payload (The checksum covers the reportID.)

7.2 SETUP_READ and READ — Reading Memory

The SETUP_READ and READ commands are used to read memory. The SETUP_READ command can be sent at anytime, but must be sent at least once before issuing any READ commands.

To read HID memory, the Bluetooth host must do the following:

1. Send a SETUP_READ command, which is a SET_REPORT transaction, to indicate the start *address* and *length* in bytes of the subsequent read.
2. Send a READ command, which is a GET_REPORT transaction, to read the data block defined by the SETUP_READ command.

Note: Additional READ commands can be sent without first sending another SETUP_READ command if reading is to take place where the last READ command left off. In other words, HID firmware will advance the start address to the *address* + *length* defined in the SETUP_READ command.

Both of these read commands will work, regardless of the storage type from which data is being read. So the commands apply whether the storage type is RAM, parallel flash, serial flash, or EEPROM. However, an indirect memory map must be used for memory peripherals, such as EEPROM and flash, that are not in the addressable space of either the CYW20730 or CYW20733.

Use the following enumeration to indirectly map EEPROM and flash memory:

```
enum
    {
        INDIRECT_MEM_MAP_MASK   = 0xFF000000,

        /// indirect memory map for EEPROM Read/Write/erase access
        INDIRECT_MEM_MAP_EEPROM = 0xFF000000,

        /// indirect memory map for parallel flash (write/erase access),
        //// read is in CPU addressable space
        INDIRECT_MEM_MAP_PF     = 0xFC000000,

        /// indirect memory map for serial flash Read/Write/Erase access
        INDIRECT_MEM_MAP_SF     = 0xF8000000,

    };
```

7.2.1 SETUP_READ

The SETUP_READ command sets the start address and length (in bytes) of a subsequent or multiple subsequent READ commands. This command can be issued by the host at anytime.

The following packet format is used with this command (using the SET_REPORT transaction):

```
struct HIDFWUReportHeader
{
    UINT8  reportID;  // report=0x71 , changeable.
    UINT32 address;
    UINT16 length;
    UINT8 checksum;
};
```

Report Data Payload (The checksum covers the payload and the reportID.)

For more information on the parameters in the `HIDFWUReportHeader` header, see Table 8.

For a sample script of the SETUP_READ command, see "SetReport".

7.2.2 READ

The READ command is used to read data from HID memory (RAM, flash, or EEPROM). Multiple READ commands can be issued after a single SETUP_READ command is sent to the HID. Each READ command sent by the host will result in a block of data being returned by the HID. The length of each block of data returned is the `length` parameter sent in the most recently received SETUP_READ command.

The following packet format is used by the host to send a read command (using the GET_REPORT transaction) to the HID:

```
struct HIDFWUReportHeader
{
    UINT8 reportID;//DEFAULT_ID_READ
    UINT8 checksum;
};
```

Report Data Payload (The checksum covers the reportID.)

The following packet format is used by the HID to send data packets (using DATA transactions) back to the host:

```
struct HIDFWUReportHeader
{
    UINT8  reportID;     //DEFAULT_ID_READ
    UINT32 address;
    UINT16 length;
    UINT8 data[length];
    UINT8 checksum;
};
```

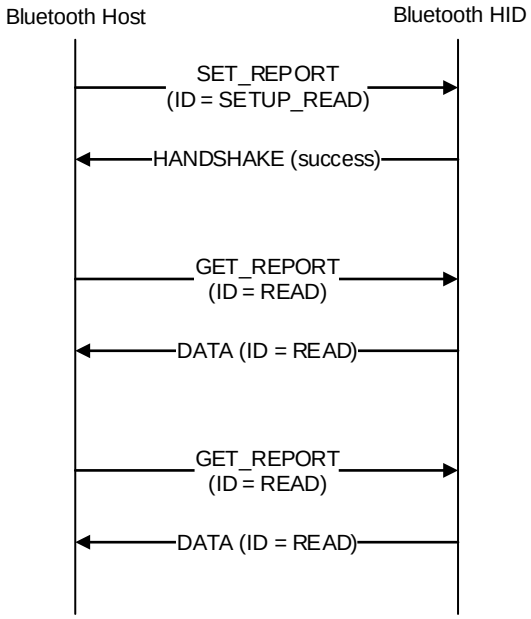
Report Data Payload (The checksum covers the payload and the reportID.)

For more information on the parameters in the `HIDFWUReportHeader` header, see Table 8.

For a sample script of the READ command, see "GetReport".

Figure 2 shows an example read sequence using some of the transaction types provided in Table 2.

Figure 2. Read Sequence



7.3 ERASE

The erase command is used to erase memory. The ENABLE_FWU command needs to be sent before erasing memory.

The ERASE command is only valid for serial flash. It does not apply to EEPROM, which does not need to be erased before overwriting. The ERASE command also does not apply to erasing parallel flash because the CYW20730 and CYW20733 only support reading from parallel flash.

The following packet format is used by the host to send this command:

```
struct HIDFWUReportHeader
{
   UINT8  reportID;      //DEFAULT_ID_ERASE
   UINT32 address;
   UINT16 length;
   UINT8 checksum;
};
```

Report Data Payload (The checksum covers the payload and the reportID.)

For more information on the parameters in the `HIDFWUReportHeader` header, see Table 8.

For a sample script of the ERASE command, see 8.3 Erasing HID Memory.

7.4 WRITE

The write command is used to write to HID memory. The ENABLE_FWU command must be sent before writing to memory.

The following packet format is used by the host to send this command:

```
struct HIDFWUReportHeader
{
    UINT8  reportID;     //DEFAULT_ID_WRITE
    UINT32 address;
    UINT16 length;
    UINT8 data[length];
    UINT8 checksum;
};
```

Report Data Payload (The checksum covers the payload and the reportID.)

For more information on the parameters in the `HIDFWUReportHeader` header, see Table 8.

For a sample script of the WRITE command, see 8.4 Writing to HID Memory.

7.5 LAUNCH

The LAUNCH command is used to indicate the address from which firmware will execute.

The following packet format is used by the host to send this command:

```
struct HIDFWUReportHeader
{
    UINT8  reportID;     //DEFAULT_ID_LAUNCH
    UINT32 address;
    UINT16 length;
    UINT8 data[length];
    UINT8 checksum;
};
```

Report Data Payload (The checksum covers the payload

The following parameters in the above `HIDFWUReportHeader` header differ from the parameters defined in Table 8:

- address: The entry address to which firmware jumps.
- If this value is 0x0000, then the host should assume that the HID will reboot.
- length: This parameter is passed to the above entry address as a parameter.
- data[]: These bytes will pass the entry address as a parameter.

8 HID Firmware Upgrade Sample Scripts

This section mostly contains reference scripts for developers of host software that is used to upgrade the firmware in a CYW20730- or CYW20733-based HID. All scripts are written in Perl.

Scripts to support the following actions are included in this section:

- Enabling a HID for a Firmware Upgrade
- Reading HID Memory
- Erasing HID Memory
- Launching Execution at a Defined HID Memory Address
- Calculating a Checksum

The following scripts, which support the above list of actions, are provided in this section:

- EnableOtafu
- ReadMem
- SetReport
- GetReport
- RxReport
- PrintReport
- EraseMem
- WriteMem
- Launch
- Checksum

8.1 Enabling a HID for a Firmware Upgrade

The following Perl subroutine is an example script for enabling a HID for a firmware upgrade:

```
#################################################################
# Enable OTAFU (send the magic 0x70 report)
#################################################################
sub EnableOtafu
{
    print "Enable OTAFU \n";
    my %command = ('Transaction Type' => 'SET_REPORT',
                   'Report Type'      => 'Feature',
                   'Report ID'        => $ID_ENABLE_OTAFU,  #0x70
                   'Data'             => "");
    BTSP::SendHIDCommand( $dutaddr, \%command );
    BTSP::WaitForSpecificEvent( $hidht, "Number Of Completed Packets" );
}
```

8.2 Reading HID Memory

The following Perl subroutine is an example script for reading memory. This Perl subroutine calls the SetReport and GetReport subroutines.

```
#################################################################
# Read memory (addr, len) -- READ SETUP set-rep then READ get-rep
#################################################################
sub ReadMem
{
    my ($addr, $len) = @_;

    SetReport($ID_SETUP_READ, $addr, $len, [ ] );
    #setup read start address and the length

    my $report = GetReport($ID_READ);

    return $report;
}
```

8.2.1 SetReport

This Perl subroutine is used to issue the SETUP_READ command.

```
#################################################################
# Send a set report( report-id, address, len, data)
#################################################################
sub SetReport
{
    my ($repid, $addr, $len, $bytes) = @_;
    my $srepid = sprintf("%02x", $repid);
    my $saddr = sprintf("%02x %02x %02x %02x", $addr&0xFF, ($addr>>8)&0xFF,
($addr>>16)&0xFF, ($addr>>24)&0xFF);
    my $sbytes = "";

    if ( scalar(@{$bytes}) > 0 ) {
        $sbytes = BTSP::ByteArrayToDataString(@{$bytes}) . " ";
    }

    my $slen = sprintf("%02x %02x", $len&0xFF, $len>>8 );
    my $checksum = Checksum($repid, $addr, $len, $bytes);
    my $schecksum = sprintf("%02x", $checksum);
    my $sreportbytes = "$saddr $slen $sbytes" . "$schecksum";

    my %command = ('Transaction Type' => 'SET_REPORT',
                   'Report Type'      => 'Feature',
                   'Report ID'        => $repid,
                   'Data'             => $sreportbytes);

    BTSP::SendHIDCommand( $dutaddr, \%command );

    RxData(); #
}
```

8.2.2 GetReport

This Perl subroutine is used to read HID memory. This section also includes the RxReport subroutine, which the GetReport subroutine calls, and the PrintReport subroutine.

```
#############################################################
# Send a GET-REPORT, return the data received
#############################################################
sub GetReport
{
    my ($reptype) = @_;

    my %command = ('Transaction Type' => 'GET_REPORT',
                   'Report Type'      => 'Feature',
                   'Report ID'        => $reptype,
                   'Size'             => 'Equal to the size of the report');

    BTSP::SendHIDCommand( $dutaddr, \%command );

    return RxReport();
}
```

8.2.2.1 RxReport

This Perl subroutine is used to receive and decode a firmware upgrade report received from the HID.

```
#############################################################
# Receive an OTAFU report, decode it into returned hash
#############################################################
sub RxReport
{
    my @report  = RxData();

    my @payloadbytes = @report[7..$#report-1];
    my $payloadref = \@payloadbytes;
    my $ret;
    my $sumpresent = $report[$#report];

    $ret->{'id'} = $report[0];

    if (length @report > 1)
    {
    $ret->{'address' }  = ($report[1] | ($report[2]<<8) | ($report[3]<<16) |
($report[4]<<24));
    $ret->{'len'     }  = ($report[5] | ($report[6]<<8));
    $ret->{'data'    }  = $payloadref;
    $ret->{'checksum'}  = $report[$#report];

    my $expected = Checksum($ret->{'id'},$ret->{'address'},$ret->{'len'},$payloadref);
    if ($sumpresent != $expected)
          {
        print "Report is ", %{$ret}, "\n";
        die sprintf("Checksums don't match: Expected 0x%02x, Got 0x%02x", $expected,
$sumpresent);
          }
    }
    return $ret;
}
```

8.2.2.2 PrintReport

This Perl subroutine is used to print a report of the hash returned by the RxReport and GetReport subroutines.

```
################################################################
# Print a report (the hash returned by RxReport/GetReport)
################################################################
sub PrintReport
{
    my ($report) = @_;
    my $bytes = $report->{'data'};

    printf("HID FWU REPORT:\n" .
           " id       = 0x%02x\n" .
           " address  = 0x%08x\n" .
           " len      = 0x%04x\n" .
           " data     = %s\n" .
           " checksum = 0x%02x\n",
           $report->{'id'},
           $report->{'address'},
           $report->{'len'},
           BTSP::ByteArrayToDataString(@{$bytes}),
           $report->{'checksum'});
}
```

8.3 Erasing HID Memory

The following Perl script is used to erase memory.

```
################################################################
# Erase memory (address, len)
################################################################
sub EraseMem
{
    my ($addr, $len) = @_;
    SetReport($ID_ERASE, $addr, $len, []);
    printf "\nErase 0x%x \n",$addr;
}
```

8.4 Writing to HID Memory

The following Perl script is used to write to memory.

```
################################################################
# Write memory (addr, bytes)
################################################################
sub WriteMem
{
    my ($addr, $bytes) = @_;
    printf "WriteMem: 0x%x, %d bytes ...\n", $addr, scalar @{$bytes};
    SetReport($ID_WRITE, $addr, scalar @{$bytes}, $bytes);
}
```

8.5 Launching Execution at a Defined HID Memory Address

The following Perl script is used to send a launch command to the HID.

```
################################################################
# Launch(address)
################################################################
sub Launch
{
    my ($addr) = @_;

    SetReport($ID_LAUNCH, $addr, 0, []);
}
```

8.6 Calculating a Checksum

The following Perl subroutine is used to compute a checksum.

```
################################################################
# Compute and return a checksum
################################################################
sub Checksum
{
    my ($id, $addr, $len, $bytes) = @_;
    my $sum = $id;

    $sum += $addr & 0xFF;
    $sum += ($addr>>8) & 0xFF;
    $sum += ($addr>>16) & 0xFF;
    $sum += ($addr>>24) & 0xFF;

    $sum += $len & 0xFF;
    $sum += ($len>>8) & 0xFF;

    my $i;
    for($i=0; $i <= $#{$bytes}; $i++)
    {
        $sum += $bytes->[$i];
    }

    return (-$sum)&0xFF; # 1-byte 2's complement of all the bytes
   }
```

8.7 Programming an Image

The following Perl subroutine is used to program (burn) an image to a HID device.

```
################################################################
# BurnImage(addr): Burn a config at the DS address passed
################################################################
sub BurnImage
{
    my ($DSaddr) = @_;
    my $line;
    my $base = 0; #$EEPROM_base_addr; # + $DSaddr;


    #printf "\nBurn image= %s ds=%x $base=%x",$burnimg,$DSaddr,$base ;

    # Parse the intel hex, writing each record to the target
    open(IHEXIN , "<$burnimg") || die "Can't open $burnimg: $!";
    LINE: while(<IHEXIN>)
```

```
    {
        /^:(..)         # $1: Record length
           (....)       # $2: Load Offset
           (..)         # $3: Record Type
           (.*)         # $4: Data (maybe empty)
           (..)         # $5: Checksum
          \s*$/x || die "Malformed line: $_";

        my $reclen  = hex($1);
        my $offset  = hex($2);
        my $rectype = hex($3);
        my $data    = $4;
        my $sum     = hex($5);

        #printf "\n offset=%x, data=%s",$offset,$data;
        my @bytes = HexStringToBytes($data);
        my $expected = Checksum($rectype,$offset,$reclen,\@bytes);
        if ($expected != $sum)
        {
            die
                sprintf("Invalid checksum: Got 0x%02X, expected 0x%02X in line\n%s",
                        $sum, $expected, $_);
        }

        if ($rectype == 0x01)     # End-of-ffile record
        {
            last LINE;
        }
        elsif ($rectype == 0x04)  # Extended linear address record
        {
            $base = hex($data);
            $base <<=16;    # upper 16 bits
           # if ($base == 0)  # we mean EEPROM
           # {
           #     $base = 0xFF000000;
            printf "\n record type =4 change base to %x\n",$base;
           # }
        }
        elsif ($rectype == 0x00)  # Data record
        {

            #WriteMem( $base | $offset , \@bytes);
            WriteMem( $base + $offset-$ds_offset_GenBurnImage +$DSaddr  , \@bytes);
            printf ".";

        }
        else # We don't support other record types
        {
            die "Unexpected record type: $rectype, in line\n$_";
        }
    }
    close(IHEXIN);
}
```

9 References

The references in this section may be used in conjunction with this document.

Cypress provides customer access to technical documentation and software through its Cypress Developer Community and Downloads and Support site (see IoT Resources).

For Cypress documents, replace the “xx” in the document number with the largest number available in the repository to ensure that you have the most current version of the document.

Document (or Item) Name	Broadcom Document Number	Cypress Document Number	Source
Cypress Documents			
Single-Chip Bluetooth Transceiver for Wireless Input Device, Datasheet	20730-DS1xx-R	002-14824, 002-15291	Cypress Developer Community
Single-Chip Bluetooth Transceiver for Wireless Input Device, Datasheet	20733-DS0xx-R	002-14859, 002-14860	Cypress Developer Community
EEPROM Format Information and Firmware Upgrading, Application Note	20730_20733-AN3xx-R		Cypress Developer Community
Application Development Kit (ADK)	–	–	Cypress representative
BlueTool™ Software	–	–	Cypress representative
Other Items			
Human Interface Device Profile 1.1	HID_SPEC_V11.pdf	–	www.bluetooth.org

Document History

Document Title: AN214821 - Upgrading Human Interface Device Firmware (Example Protocol, Commands, and Scripts)

Document Number: 002-14821

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	–	–	04/08/2013	20730_20733-AN400-R Initial release
*A	5473869	UTSV	10/14/2016	Updated to Cypress template. Added Cypress Part Numbering Scheme.
*B	5836682	MALI	07/27/2017	Migrated to the new MS Word template. Removed references to Broadcom wherever applicable

Worldwide Sales and Design Support

Cypress maintains a worldwide network of offices, solution centers, manufacturer's representatives, and distributors. To find the office closest to you, visit us at Cypress Locations.

Products

ARM® Cortex® Microcontrollers	cypress.com/arm
Automotive	cypress.com/automotive
Clocks & Buffers	cypress.com/clocks
Interface	cypress.com/interface
Internet of Things	cypress.com/iot
Memory	cypress.com/memory
Microcontrollers	cypress.com/mcu
PSoC	cypress.com/psoc
Power Management ICs	cypress.com/pmic
Touch Sensing	cypress.com/touch
USB Controllers	cypress.com/usb
Wireless Connectivity	cypress.com/wireless

PSoC® Solutions

PSoC 1 | PSoC 3 | PSoC 4 | PSoC 5LP | PSoC 6

Cypress Developer Community

Forums | WICED IOT Forums | Projects | Videos | Blogs | Training | Components

Technical Support

cypress.com/support

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709